International Journal of Computer Science Engineering
and Information Technology Research (IJCSEITR)
ISSN (P): 2249-6831; ISSN (E): 2249-7943
Vol. 9, Issue 2, Dec 2019, 7-14

FAULT TOLERANCE IN DEFENCE C4I SYSTEMS

SANDEEP KUMAR & MANOJ TYAGI

Bharat Electronics Limited, Ghaziabad, Uttar Pradesh, India

ABSTRACT

Over the last few years, Fault Tolerance has become a major requirement for stability across variety of systems. Defence C4I systems provide Commanders/Captains capability of situational awareness; information about the location and status of enemy and friendly forces and support quick and accurate tactical decision making. Thus Fault-tolerance is particularly sought-after in such life-critical defence C4I systems demanding high-availability. A small delay can result in fatal losses as defence systems are real time scenario based systems. So there is a need for efficient fault tolerance functionality in critical systems. Fault-tolerance enables C4I system to continue operating properly in the event of the failure of some of its components. This paper mainly focuses on the need, methodologies and strategies for Fault Tolerance in Army/Air Force/Navy C4I Systems.

KEYWORDS: Defence C4I, Fault Tolerance & Switching Policy

Received: May 07, 2019; **Accepted**: May 28, 2019; **Published**: Jun 14, 2019; **Paper Id.**: IJCSEITRDEC20192

Original Article

INTRODUCTION

Defence C4I (command, control, communication, computer and intelligence) are complex information systems designed to achieve mission objectives through better situational awareness and information superiority. System is the entire set of components, both computer related, and non-computer related to achieve specific mission objectives.

Fault-tolerance or graceful degradation is the property that helps defence C4I systems to continue operating properly in the event of the failure of some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naïvely-designed system in which even a small failure can cause total breakdown.

The step-by-step approach to achieve fault tolerance is explained in below algorithm

Algorithm

- *FTS monitors for faults*
- *Fault Occurs*
- *FTS detects fault*
- *FTS analyzes fault*
- *FTS takes corrective action*

There is a difference between fault tolerance and systems that rarely have problems. For instance, the Western Electric crossbar systems had failure rates of two hours per forty years, and therefore were highly fault

resistant. But when a fault did occur they still stopped operating completely, and therefore were not fault tolerant [1].

LEVELS OF FAULT TOLERANCE

There are three levels at which fault tolerance can be applied [2].

Level 1

Hardware fault tolerance measures include redundant communications, replicated processors, additional memory, and redundant power/energy supplies. Hardware fault tolerance was particularly important in the early days of computing, when the time between machine failures was measured in minutes.

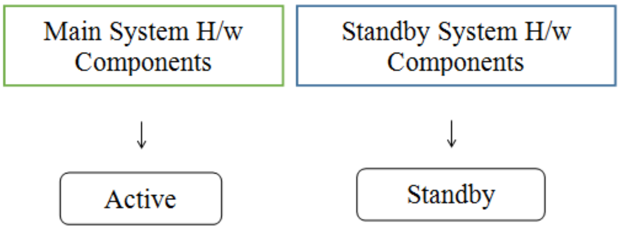

Figure 1: Level 1 H/w Fault Tolerance

Level 2

A second level of fault tolerance recognizes that a fault tolerant hardware platform does not, in itself, guarantee high availability to the system operator. It is still important to structure the computer software to compensate for faults such as changes in program or data structures due to transients or design errors. This is software fault tolerance. Mechanisms such as check- point/restart, recovery blocks and multiple-version programs are often used.

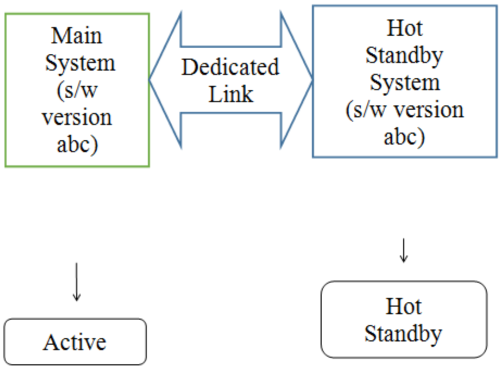

Figure 2: Level 2 S/w Fault Tolerance

Level 3

At a third level, the computer subsystem may provide functions that compensate for failures in other army/airforce/navy system facilities that are not computer-based. This is system fault tolerance. For example, software can detect and compensate for failures in sensors. Measures at this level are usually application-specific. It is important that fault tolerance measures at all levels be compatible, hence the focus on system-level issues, leveraging manpower and engineering capability within the country for attaining self-reliance in design, development, and manufacturing in the defence sector.

HOW FAULT TOLERANCE IS ACHIEVED

The Fault Tolerance Software provides an assessment of the availability/ non-availability of the nodes and initiates appropriate configured action on hardware/ software level fault detection. This tolerance is achieved by utilizing the services of redundant hardware at both intra-node and inter-node level.

Arbitrate Between Nodes

This feature enables the FTS to decide which one of the two nodes shall be Active at the time of system startup.

Algorithm

- *Node Powers Up*
- *Node waits for the Arbitration/ health message from other node*
- *Does Node get the message?*
- *If Yes, go to iv), else go to v)*
- *Node configures itself as Dual Node Self Hot-Standby or Dual Node Self Active*
- *Send message on Backup Link*
- *Is message received?*
- *If Yes, go to iv), else go to v)*
- *Node configures itself as Hot-Standby*
- *Node configures itself as Single Node Active*

Node Status Management (Inter Node Failure)

This function shall enable the program to keep track of the health of other node. This shall be done to detect whether the other node has failed and ensure the recovery in case of failure of other node.

Node Switch Over

Algorithm

- *Nodes are in Active/ Hot-Standby Configuration*
- *Has Active node stopped responding?*
- *If yes, go to iii), else stay on ii)*
- *Hot-Standby Node reconfigures itself as Active*
- *Start the node output to other segments*

Start the node output to other segments. This function enables the system to continue in working mode in the event of failure/performance degradation of the Active Server.

Reintegration

This feature enables the program to bring the currently switched on node to the level of active node, and make it Hot-Standby node.

Algorithm

- *Node Powers Up*
- *Node waits for the health message from other node*
- *Does Node get health message?*
- *If Yes, go to iv), else go to ix)*
- *Is Arbitration decision to make node Active?*
- *If Yes, go to v), else go to vi)*
- *Node configures itself as Dual Node Self Active*
- *Node configures itself as Dual Node Self Hot-Standby*
- *Request Checkpoint from Active Node*
- *On receiving checkpoint, configure the CSCIs*
- *Is message received?*
- *If Yes, go to x), else go to xi)*
- *Node configures itself as Hot-Standby*
- *Node configures itself as Single Node Active*

Check Pointing

Check pointing is a mechanism to record the state of a program under execution. This state is then used to populate a fresh application in order to bring the new application to the current state with respect to the check pointed application

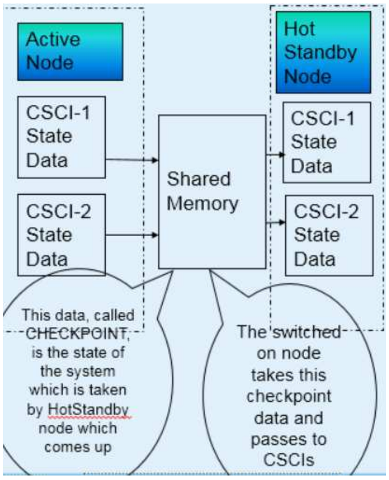

Figure 3: Check Pointing

SWITCHING POLICIES FOR FAULT TOLERANCE

FTS system monitors the Temperature, CPU Usage, RAM Usage, Status of Process and Responsiveness of Process. The Switching Action is described in tables (Table 1 to Table 5) below for each of these cases.

The Table S1 explains that if for one of the systems, temperature goes beyond limits, then the other available system shall take over.

Table 1: System Temperature

S1 (Active)	S2 (Standby)	Switching Policy
Normal	Normal	No Action
Overheat	Overheat	No Action
Normal	Overheat	No Action
Overheat	Normal	->S2 shall become Active ->S1 shall become Standby

Table 2: System CPU Usage

S1 (Active)	S2(Standby)	Switching Policy
Within CPU usage limit	Within CPU usage limit	No Action
Abnormal	Abnormal	No Action
Within CPU usage limit	Abnormal	No Action
Abnormal	Within CPU usage limit	->S2 shall become Active ->S1 shall become Standby

Table 3: System RAM Usage

S1 (Active)	S2(Standby)	Switching Policy
Within RAM usage limit	Within CPU usage limit	No Action
Abnormal	Abnormal	No Action
Within RAM usage limit	Abnormal	No Action
Abnormal	Within RAM usage limit	->S2 shall become Active ->S1 shall become Standby

Table 4: CSCI Process Status

S1 (Active)	S2 (Standby)	Switching Policy
Running	Running	No Action
Crash	Crash	No Action
Normal	Crash	No Action
Crash	Running	->S2 shall become Active ->S1 shall become Passive

Table 5: CSCI Responsiveness Status

S1 (Active)	S2 (Standby)	Switching Policy
Responsive	Responsive	No Action
Non-responsive	Non-responsive	No Action
Responsive	Non-responsive	No Action
Non-responsive	Responsive	->S2 shall become Active ->S1 shall become Passive

MANUAL CONTROL USING CLI

Command Line Interface (CLI) is present on both servers.

In case of CLI command to exchange servers, CLI sends the command to FTS. FTS changes currently Active

Server to Standby and currently Standby Server to Active.

Graceful shutdown of Standby Server

In case of CLI command for graceful shutdown, CLI sends shutdown command to FTS. On receiving this command, FTS informs other node FTS and kills itself after releasing all resources captured by FTS. FTS on other node changes state to SINGLE_NODE_ACTIVE

EXPERIMENTAL DETAILS

- Two nodes (Node 1 and Node 2), each replica of each other in terms of hardware and software were made available.
- Node 1 was turned on and Node 2 was kept off.
- After Node 1 had booted up, Node 2 was turned on
- Time taken by Node 2 to become a replica of Node 1 was noted.

It was found that:

- Time taken by Node 2 to become a replica of Node 1 was proportional to the checkpoint data maintained by CSCIs running on Node 1.

Afterwards, Node 1 was powered off and time taken by Node 2 to take over was noted.

It was found that

- Time taken by Node 2 is independent of checkpoint data of Node 1. It was further observed that the switchover time depends on number of external links with System (of Node 1/Node 2) and communication architecture, including the heartbeat rate.

HA AND FTS

High-availability clusters (also known as HA clusters or fail-over clusters) are groups of computers that support server applications that can be reliably utilized with a minimum amount of down-time. They operate by using high availability software to harness redundant computers in groups or clusters that provide continued service when system components fail. Without clustering, if a server running a particular application crashes, the application will be unavailable until the crashed server is fixed. HA clustering remedies this situation by detecting hardware/software faults, and immediately restarting the application on another system without requiring administrative intervention, a process known as failover[3].

As part of this process, clustering software may configure the node before starting the application on it. For example, appropriate file systems may need to be imported and mounted, network hardware may have to be configured, and some supporting applications may need to be running as well[3].

Fail-safe architectures may encompass also the computer software, for example by process replication [4].

High availability, in general, is an excellent part of any large scale disaster recovery strategy [5].

In case of any of the following faults, clustering shall switch over to other node:

- Any spawned service/application gets killed.
- Node gets disconnected from the network due to any reason.
- Node gets powered-off.

During switchover, clustering shall kill all the spawned services/applications running on current node and spawn them on other node.

Fault Tolerant Hardware and Software Solutions both provide extremely high level of availability, there is a trade-off: Fault-Tolerant Servers achieve HA with a minimum amount of system overhead to deliver a superior level of performance, while Fault-Tolerant Software can be run on industry-standard servers which might already be a part of system.

DDS AND FTS

In DDS, there is a native feature called exclusive ownership. Normally, DDS allows multiple publishers to update the same data-item simultaneously. This behavior is called shared ownership and it is the default setting for the ownership Quality of Service (QoS) setting. If the non-default setting, called exclusive ownership, is selected, the infrastructure will make one of the publishers the owner of the data-item -- at any time, only the owner of the data-item can update its state. For this leader selection, another policy called ownership strength can be adjusted. This integer value will be inspected by the middleware to identify the current leader at any time that is the publisher with the highest ownership strength. This selection is done consistently throughout the whole system[6].

Other than selecting the right QoS, this solution is transparent to all components in the system. No extra code has to be added, nor tested, nor maintained. For optimal robustness, implementations do send data over the wire for each of the active publishers, and the leader selection takes place at the receiver side. This means that the feature does result in extra networking traffic.

CHALLENGES IN FTS

- To keep on reducing the time taken for switchover
- To make checkpoint process transparent to application
- To keep on reducing the size of checkpoint data

CONCLUSIONS

The paper has explained that different scenarios need different levels of Fault tolerance. Various techniques to achieve fault tolerance for various levels have been detailed. As evident, time duration involved in achieving seamless working systems is also of due importance and the paper has successfully explained the factors controlling this time duration. Also various environment/system parameters have been detailed which can be used to chose one of the redundant systems providing fault tolerance as Primary. Paper dwells into the Available Fault Tolerance techniques like High Availability and DDS, which are used widely.

Fault Tolerance Systems are in infant stage. A shift in paradigm is needed from graceful degradation to a system which is 100 percent transparent to any kind of system hiccups so that user experiences a seamless stable system.

REFERENCES

1. *Fault tolerance, https://en.wikipedia.org/wiki/Fault_tolerance*
2. *Walter L. Heimerdinger, Charles B. Weinstock, A Conceptual Framework for System Fault Tolerance, Technical Report, CMU/SEI-92-TR-033,ESC-TR-92-033,Oct 1992*
3. *High Availability cluster, https://en.wikipedia.org/wiki/High-availability_cluster*
4. *Chandra Guru, B. P., & M S, A. (2018). Innovative Defence Management by Tipu Sultan.*
5. *Fault Tolerance, https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Fault_tolerance.html*
6. *David Davis, Understanding VMware Fault Tolerance benefits and requirements, https://searchservervirtualization.techtarget.com/tip/Understanding-VMware-Fault-Tolerance-benefits-and-requirements*
7. *Reinier Torenbeek, Achieving fault tolerance by replicating Data Writers,, October 30, 2012, https://www.rti.com/blog/2012/10/30/achieving-fault-tolerance-by-replicating-datawriters/*

ABOUT THE AUTHORS

Mr Manoj Tyagi has obtained his MSc (Defence Simulation & Modelling) from Royal Military College of Science, Cranfield University, Shrivenham, UK and MS (Software Systems) from BITS Pilani. He has received many rewards, including “DRDO Lab Technology Award” and “DRDO Lab Scientist of the year Award”. His interest includes Defence Experimentation and AI based Application Development.

Mr Sandeep Kumar obtained his B.E. (Comp Sc & Engg) from Panjab Engineering College, Chandigarh in 2002. . He has worked in HCL in embedded systems and consumer electronics. He has extensively worked in Surveillance systems for Army/Navy/AirForce in Bharat Electronics Ltd. His interests include Embedded Systems and Artificial Intelligence.